Express Mail Label No. EL889676255US

METHOD FOR PROVIDING CONCURRENT NON-BLOCKING HEAP MEMORY MANAGEMENT FOR FIXED SIZED BLOCKS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not applicable.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The preferred embodiments of the present invention are directed to run time management of heap memory. More particularly, the preferred embodiments of the present invention are directed to a concurrent non-blocking heap memory management method that allows software to remove and return blocks of memory from the heap simultaneously with a hardware agent returning blocks to the heap.

Background of the Invention

[0004] In the art of computer programming, a programmer may not know at the time of coding the amount of memory required to perform a particular operation. Rather than statically allocate memory large enough to encompass any situation that may arise, programmers may dynamically allocate memory at run time necessary to perform the desired operation, thus improving the utilization of computer resources.

[0005] Memory allocated for use at run time is typically referred to as heap memory. Heap memory is allocated for use by a particular process, which may include multiple threads. This use typically comprises the one or more software threads claiming or removing blocks of the heap memory, using the blocks of heap memory, and then returning the blocks to the unused heap pile for removal and use by other software threads.

[0006] An exemplary use of a removed block of heap memory is a buffer for the exchange of command lists and/or data from software threads to hardware devices. That is, a software thread may need to program or pass large amounts of data to a hardware device, and the size of the program or data block may be too large to pass by way of a direct communication message. In such an situation, the related art software threads claim or remove a portion of heap memory (which may include one of more blocks), place the command lists and/or data into the memory locations, and inform the hardware device of the location in main memory of the command lists and/or data locations. Once the hardware completes the necessary tasks or reads the data, the heap memory block or blocks remain removed from the unused heap pile.

[0007] In related art computer systems, the method by which blocks of heap memory are returned after a hardware device completes its tasks is by a software thread, either the invoking thread or another software thread, returning the block to the heap pile. More particularly, in related art computer systems, the hardware device invokes an interrupt to the microprocessor, which preempts executing software streams and loads and executes an interrupt service routine. The interrupt service routine identifies the reason for the interrupt, which is the notification that the hardware task has completed and the heap memory block or blocks are no longer needed, and either returns the heap memory block, or invokes other software streams to return the memory

block. Thus, a software stream returns the block to the heap memory for further claiming or removal.

[0008] Returning heap memory using interrupts could be inefficient. This inefficiency is seen not only in the use of an interrupt from the hardware device to the microprocessor to pass the message that the heap memory block may be returned, but also in preempting other software streams to service the interrupt and return the block.

[0009] Thus, what is needed in the art is a way to return blocks of heap memory that does not require assistance of the central processing unit or software streams.

BRIEF SUMMARY OF THE INVENTION

[0010] The problems noted above are solved in large part by a run time heap memory management method and related system that allows a hardware device, or an agent for hardware, to return heap memory blocks to the unused heap pile without intervention from the calling software stream, an interrupt service routine, or the like. The preferred implementation is a heap memory management method that works as a modified stack structure. Software preferably removes heap memory blocks and replaces heap memory blocks to the heap pile in a last-in/first-out (LIFO) fashion. A hardware device preferably returns heap memory blocks to the heap pile at the end or bottom of the stack without intervention of the software that removed the block of heap memory.

[0011] More particularly, the heap memory management method of the preferred embodiments comprises managing the blocks of the heap memory in a linked list format, with each memory block in the heap pile identifying the next unused block. Thus, removal of a heap memory block by a software stream preferably involves changing the value of a top pointer register, freeing the heap memory block previously listed in the top pointer register for use. Likewise, returning a block of heap memory to the heap pile by software streams preferably involves changing the

address of the top pointer, and writing a portion of the heap memory block to be returned to link or point to the next block of memory in the list. While software streams remove and replace blocks of heap memory to the top of the list in a LIFO fashion, preferably hardware returns heap memory blocks to the bottom or end of the list by writing a null in the next block field of the block to be returned, changing the next block field of the last entry to point to the block to be returned, and updating a bottom pointer register to point to the block to be returned.

[0012] In the preferred implementation, however, one block of heap memory, with its next block field indicating a null, remains in the list and cannot be removed even if all the remaining heap memory blocks are removed. A hardware device, or an agent for multiple hardware devices, thus always has the capability of placing blocks of heap memory back in the heap pile.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] For a detailed description of the preferred embodiments of the invention, reference will now be made to the accompanying drawings in which:

[0014] Figure 1 shows a computer system constructed in accordance with the preferred embodiment;

[0015] Figure 2A shows an exemplary heap of memory comprising a plurality of fixed size blocks;

[0016] Figure 2B shows the exemplary heap memory of Figure 2A in a more graphical form;

[0017] Figure 3 shows a high level flow diagram of the steps to initialize the heap memory;

[0018] Figure 4A shows the exemplary linked list of Figure 2B with the two uppermost blocks removed;

[0019] Figure 4B shows the exemplary linked list of Figure 4A with heap memory block zero returned to the top of the list;

[0020] Figure 5 shows a high level flow diagram of the steps used by software streams to remove blocks of heap memory;

[0021] Figure 6 shows a high level flow diagram of the steps used to return blocks of heap memory by software streams;

[0022] Figure 7A shows the exemplary linked list with all but the last block removed;

[0023] Figure 7B shows the return by hardware of a block of heap memory to the exemplary linked list of Figure 7A; and

[0024] Figure 8 shows a high level flow diagram of the return of blocks of heap memory by hardware devices.

NOTATION AND NOMENCLATURE

[0025] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, computer companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function.

[0026] In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to...". Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

[0027] In this specification, and in the claims, the term "heap memory" refers generally to memory that is allocated to a software stream or streams for use during run time. The term "heap pile" refers to blocks of heap memory that have not been removed for use from the linked list of

available blocks. Thus, to return a block of heap memory to the heap pile is to return the block of heap memory to the linked list such that it may be removed again at a later time.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0028] Figure 1 shows a computer system 100 constructed in accordance with the preferred embodiment. Computer system 100 generally comprises a microprocessor or CPU 20 coupled to a main memory array 26, and various other peripheral computer system components, through an integrated host bridge 22. The CPU 20 preferably couples to the host bridge 22 via a host bus 24, or the host bridge logic 22 may be integrated into the CPU 20. The CPU 20 may comprise any microprocessor commercially available, for example, a Pentium® III or a Pentium® IV made by Intel Corporation. Though only one microprocessor is shown in Figure 1, the preferred embodiments find application in multiple microprocessor systems, which may likewise contain several host bridge devices, each coupled to a plurality of microprocessors.

[0029] Main memory array 26 preferably couples to the host bridge 22 through a memory bus 28. The host bridge 22 preferably includes a memory control unit (not shown) that controls transactions to the main memory 26 by asserting necessary control signals during memory accesses. The main memory 26 functions as the working memory for the CPU 20, and any additional microprocessors coupled to the host bridge 22. Generally, the main memory array 26 comprises a conventional memory device or array of memory devices in which programs, instructions and data are stored. The main memory array 26 may comprise any suitable type of memory such as dynamic random access memory (DRAM) or any of the various types of DRAM devices such as synchronous DRAM (SDRAM), extended data output DRAM (EDO DRAM), or Rambus™ DRAM (RDRAM).

[0030] The computer system 100 also preferably comprises a graphics controller or video driver card 30 that couples to the host bridge 22 by way of bus 32, which bus could be an Advanced Graphics Port (AGP), or other suitable bus. Alternatively, the graphics controller may couple to the primary expansion bus 34 or one of the secondary expansion buses, for example, peripheral component interconnect (PCI) bus 40. Graphics controller 30 further couples to a display device 36, which may comprise any suitable electronic display device upon which any image or text can be represented.

[0031] The computer system 100 also preferably comprises a second bridge logic device, input/output (I/O) bridge 38, that bridges the primary expansion bus 34 to various secondary buses including a low pin count (LPC) bus 42 and the PCI bus 40. The bridge device 36 may be any suitable bridge device on the market. Although the I/O bridge 38 is shown in Figure 1 only to support the LPC bus 38 and PCI bus 40, various other secondary buses may be supported by the I/O bridge 38.

[0032] The primary expansion bus 34 may comprise any suitable expansion bus. If the I/O bridge 38 is an ICH 82801AA made by Intel Corporation, then the primary expansion bus may comprise a Hub-link bus, which is a proprietary bus of Intel Corporation. However, computer system 100 is not limited to any particular type of primary expansion bus, and thus other suitable buses may be used, for example, a PCI bus.

[0033] Figure 1 also shows a read only memory (ROM) device 44 coupled to the I/O bridge 38 by way of the LPC bus 42. The ROM 46 preferably contains software programs executable by the CPU 20 which may comprise programs to implement basic input/output system (BIOS) commands, and also instructions executed during and just after power on self test (POST) procedures. The LPC bus 42 also couples a super I/O controller 46 to the I/O bridge 38. The super

I/O controller 46 controls many computer system functions including interfacing with various input and output devices such as keyboard 48. The super I/O controller 48 may further interface, for example, with a system pointing device such as mouse (not shown), various serial ports (not shown) and floppy drives (not shown). The super I/O controller 48 is often referred to as "super" because of the many I/O functions it may perform.

[0034] Figure 1 further shows a portion of the main memory array 26 is preferably designated as heap memory 50. Figure 2A shows the heap memory 50 in block diagram form. Preferably, main memory allocated for heap memory purposes is a set of continuously addressed memory locations, and the heap memory is divided into fixed size blocks -- each block preferably having the same number of bytes, and the number of bytes being a 2^N multiple (1, 2, 4, 8, 16, 32, 64, etc.). In this way, the address of a block within the heap memory 50 is a shift operation based on the base address 52, preferably the first memory location, the number of blocks and the size of each block.

[0035] The exemplary heap memory in Figure 2A comprises eight fixed sized blocks (labeled 0 through 7 in the figure). Each block, while stored unused in the heap pile, preferably has a next block field 54 that points to the next block in the linked list, preferably by block number. Thus, block 0 of Figure 2A has a next block field 54A that points or links to block 1, block 1 has a next block field 54B that links to block 2, and so on through block 6 having a next block field 54G that links to block 7. Block 7, being the last block in the exemplary linked list, has a next block field 54H content of null. The importance of the last block is discussed more thoroughly below. Figure 2B shows much the same information as Figure 2A, but better exemplifies the preferred linked list orientation of the blocks. In particular, Figure 2B shows that the next block field 54A of block 0 points to block 1. The next state field 54B of block 1 points to block 2, and so on. Figure 2B also shows two registers, Top register 56 and Bottom register 58. Preferably, Top

register 56 points to the first or top block of the heap memory in the linked list, in the exemplary case of Figure 2B block 0. Likewise, Bottom register 58 preferably points to the bottom end or last block of heap memory in the linked list, in the exemplary case of Figure 2B block 7. It must be understood however that the linked list shown in Figures 2A and 2B are merely exemplary. The number of blocks in the heap memory is subject to change at the whim of the software stream that creates the heap, and the particular order of the linked blocks changes as those blocks are removed and returned from the list.

[0036] Figure 3 shows a high level flow diagram of the steps to initialize the heap memory for use in the preferred embodiments, starting with step 60. In calling a software routine implementing the steps of Figure 3, preferably three parameters are passed by the calling routine: the Base Memory Address of the heap, the Heap Size, and a preferred Block size. It must be understood that in the preferred embodiments of the present invention no particular heap size or block size is more preferred. These parameters are determined by the calling routine.

[0037] After the software routine is called and the appropriate parameters passed, preferably all the entries in the heap memory are cleared (set to zeros) (step 62). Thereafter, the next block field 54 of each block is initialized to form the linked list (step 64), which may initially appear similar to the linked lists shown in Figures 2A and 2B. After initializing of the next block fields, the number of the first block in the linked list is inserted into the Top register 56 (step 66), and the number of the last block in the list is inserted in the Bottom register 58 (step 68). Finally, the routine ends (step 70) by returning a Heap_Handle to the calling software, which Heap_Handle comprises pointers to the Top and Bottom registers, as well as the base addresses of the heap memory.

[0038] After the heap has been allocated and initialized into the linked list structure of the preferred embodiments, software streams are free to remove blocks from the list for use. In broad terms, removal of the blocks from the heap memory by software streams is preferably a last-in/first-out (LIFO) scheme, also known as a stack. Blocks of heap memory are preferably removed from the top or beginning of the list, and they are preferably returned by software streams to the top of the list. Figure 4A shows the exemplary linked list of Figure 2B with the two upper most blocks (blocks 0 and 1) removed from the list. What remains in Figure 4A, of the exemplary eight blocks shown in Figure 2B, are blocks 2 through 7.

[0039] Figure 5 shows a high level flow diagram of the steps preferably used by software streams to remove blocks of heap memory from the heap pile. In particular, the routine preferably starts at step 72, and is passed the Heap_Handle parameter. The first step preferably involves reading the number held in the Top Register 56, to determine the next or top block on the linked list that can be removed (step 74). Next, the top block's next block field 54 is read (step 76), and if the next block field of the top block is null, then the list is considered empty and the routine ends returning an empty list indicator (steps 78 and 84). The importance of indicating an empty list when a single block remains is discussed with respect to a hardware device or an agent for multiple hardware devices returning blocks of heap memory to the heap pile. If the next block field of the top block of the linked list contains a block number, and not a null character, the block number in the next block field of the top block is preferably atomically written to the Top register 56 (step 80). By writing the Top register 56, the block pointed to by the Top register becomes the next available block at the top of the heap memory. One of ordinary skill in the art understands that step 80 of Figure 5 inherently contains additional steps required in a shared variable environment. In particular, writes to shared memory locations require atomic operations such as

atomic *compare-and-swap* primitives which also implement retries when the compare step fails. For a more detailed discussion of concurrent non-blocking algorithms, refer to "Non-Blocking Algorithms and Preemption-Safe Locking on Multipleprogrammed Shared Memory Multiprocessors," authored by Maged M. Michael and Michael L. Scott, Journal of Parallel and Distributed Computing, 1998, pages 1-26, which paper is incorporated by reference as if reproduced in full below. Finally, the address of the block removed is calculated (step 82), and the process returns a pointer to the removed block (step 84).

[0040] Because access to the heap memory is preferably non-blocking, a software stream attempting to remove the block preferably does not block access by other software streams to parameters such as the Top register. For this reason, it is possible that the value of the Top register may change between the step of reading the block number of the first block on the list (step 74), and writing a new block number to the Top register (step 80). In such a circumstance, the process of removal preferably starts anew at step 72. One having ordinary skill in the art understands the limitations of concurrent non-blocking algorithms, and now understanding the steps involved in the removal process could account for the contingencies in a software program implementing the steps of Figure 5.

[0041] Because in the preferred embodiments the blocks are of fixed 2^N size, calculating the base address of the removed block is a shift operation of the base memory address of the heap. Consider an exemplary case where the block size is eight bytes, implying a difference in starting addresses between contiguous blocks of three bits. In this exemplary case, determining the address of any removed block involves shifting the block number by three bits, and adding the shifted result to the base address. Preferably, the base memory address has the lowest address value, with the heap memory addresses growing larger toward the end of the heap. Thus, if block

0 is removed, calculating the address of the first memory location of block 0 simply involves shifting 0 (which is still zero) and adding the shifted result to the base memory address. This is consistent with block 0 being the first block in the heap memory. If, however, the removed block of heap memory is block 2 (10 binary), calculating the starting address involves shifting the block number three bits and adding the shifted result to the base memory address.. Thus, the shift operation is left shift; however, it is equally valid to have the base address of the heap memory as the largest address, and in this case shifts to determine addresses of the block need to be a right shift (or division operation). The process exemplified in Figure 5 is for removal of a single block. If multiple blocks are required, the process is preferably repeated multiple times.

[0042] Consider now the adding or return of a block of heap memory to the unused heap pile by a software stream where, prior to the return, the linked list is as exemplified in Figure 4A. Figure 4B shows that, in broad terms, when software returns blocks of heap memory, in this exemplary case block 0, the returned blocks are preferably placed at the beginning of the linked list, consistent with preferred stack operation heap memory management.

[0043] Figure 6 shows a high level flow diagram detailing the return of blocks to the unused heap pile by software streams. Preferably after starting (step 86) and passing of the Heap-Handle and block number parameters, the first step is a read of the Top register 56 to determine the top block in the linked list (step 88). Once the top block has been determined, preferably the top block number is written to the next block field of the block to be returned (step 90). Once the link is made by having the next state field of the block to be returned reflecting the top block, the returned block is effectively added to the list by atomically writing the returned block number to the Top register 56 (step 92) and the process ends (step 94). Again, one of ordinary skill in the art

understands that the steps exemplified by step 90 require atomic *compare-and-swap* primitives, or the like, which implement the concurrent non-blocking aspect of the preferred embodiments.

[0044] Summarizing before continuing, software streams remove memory blocks from the heap by taking the first block in the linked list and atomically updating the Top register 56. Likewise, software streams return blocks of heap memory by updating the next block field of the block to be returned to point to the first block of the free list, then atomically writing the Top register 56 to point to the returned block. Thus, as for software removal and return of a block of heap memory, the linked list works in a LIFO fashion. In the preferred embodiments however, a hardware device, or an agent for multiple hardware devices, has the ability to return blocks of heap memory to the heap pile.

[0045] Figure 7A shows the linked list of the exemplary heap memory with all but the seventh block removed. In the preferred embodiments, one block remains in the linked list when the list is considered empty. Thus, if a calling routine attempts to remove a block from heap memory with only one block remaining, the calling routine is preferably returned an indication that no blocks are available (steps 78 and 84 of Figure 5). By requiring at least one block of heap memory in the linked list at all times, a hardware agent may return blocks without causing contention with software streams. More particularly, a hardware device, or an agent for multiple hardware devices, preferably returns blocks to the linked list at the bottom or end of the list. Thus, while software operates on a LIFO basis at the top of the stack, hardware returns blocks to the bottom. Consider an exemplary return of block 0 by a hardware agent to the linked list shown in Figure 7A. Figure 7B shows the status of the linked list after the return of the block 0 by a hardware agent.

[0046] Figure 8 shows a high level flow diagram of the steps to return a block of heap memory to the unused heap pile by hardware devices, starting with the passing of the Heap-Handle and

block number parameters at step 96. The first step is to write a null to the next state field of the block to return to the heap pile (step 98). Next, the Bottom register 58 is read (step 102), thus identifying the block of heap memory occupying bottom or end position of the linked list. The Bottom register 58 is updated with the number of the block to return (step 104), and the current last block's next block field is updated to contain the block number of the block to return (step 106). Thus, the process of returning a block of heap memory to the heap pile is complete, and the process returns (step 108).

[0047] In the preferred embodiment, only a single agent is allowed to return blocks of heap memory in the fashion described. The single agent could be a hardware device, or could be an agent acting on behalf of multiple hardware devices. A non-limiting list of hardware devices that could implement the heap memory management method comprises graphics cards, network interface cards, audio devices, and mass storage devices such as hard drives and compact disc drives.

[0048] By allowing software threads or streams to operate on a first end of the linked list, and hardware devices to operate on a second end of the linked list, the return and removal process may take place simultaneously. In having at least one block of heap memory in the linked list when the list is considered empty, software streams and hardware devices need not access the same registers, thereby avoiding contention. That is, software streams need only access the Top register 56 for both removal and return of blocks, and hardware need only access the Bottom register 58.

[0049] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.